International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Z.130

Amendment 1

(06/2006)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

Formal description techniques (FDT) – Extended Object Definition Language (eODL)

Extended Object Definition Language (eODL): Techniques for distributed software component development – Conceptual foundation, notations and technology mappings

Amendment 1: New Annex E – eODL to CIDL mapping

ITU-T Recommendation Z.130 (2003) – Amendment 1

ITU-T Z-SERIES RECOMMENDATIONS

LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

FORMAL DESCRIPTION TECHNIQUES (FDT)	
Specification and Description Language (SDL)	Z.100–Z.109
Application of formal description techniques	Z.110–Z.119
Message Sequence Chart (MSC)	Z.120–Z.129
Extended Object Definition Language (eODL)	**Z.130–Z.139**
Testing and Test Control Notation (TTCN)	Z.140–Z.149
User Requirements Notation (URN)	Z.150–Z.159
PROGRAMMING LANGUAGES	
CHILL: The ITU-T high level language	Z.200–Z.209
MAN-MACHINE LANGUAGE	
General principles	Z.300–Z.309
Basic syntax and dialogue procedures	Z.310–Z.319
Extended MML for visual display terminals	Z.320–Z.329
Specification of the man-machine interface	Z.330–Z.349
Data-oriented human-machine interfaces	Z.350–Z.359
Human-machine interfaces for the management of telecommunications networks	Z.360–Z.379
QUALITY	
Quality of telecommunication software	Z.400–Z.409
Quality aspects of protocol-related Recommendations	Z.450–Z.459
METHODS	
Methods for validation and testing	Z.500–Z.519
MIDDLEWARE	
Distributed processing environment	Z.600–Z.609

For further details, please refer to the list of ITU-T Recommendations.

ITU-T Recommendation Z.130

Extended Object Definition Language (eODL): Techniques for distributed software component development – Conceptual foundation, notations and technology mappings

Amendment 1

New Annex E – eODL to CIDL mapping

Summary

This amendment provides an example mapping of ITU eODL for technology independent component specifications into a technology dependent one, which in this case is the CIDL (OMG Component Implementation Definition Language as part of CORBA 3.0). This amendment transforms (by means of different mappings) the concept of components from design and implementation (where modules are well known) to binary software. The composition of components takes place during execution time.

Source

Amendment 1 to ITU-T Recommendation Z.130 (2003) was approved on 13 June 2006 by ITU-T Study Group 17 (2005-2008) under the ITU-T Recommendation A.8 procedure.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure e.g. interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

CONTENTS

		Page
1)	Replace the following part in the items of Summary:	1
2)	Update the Contents as follows:	1
3)	Insert before Appendix I:	1

ITU-T Recommendation Z.130

Extended Object Definition Language (eODL): Techniques for distributed software component development – Conceptual foundation, notations and technology mappings

Amendment 1

New Annex E – eODL to CIDL mapping

1) Replace the following part in the items of Summary

- Annex D contains a software reference to the XML representation [12] of the eODL metamodel according to the XML meta interchange format (XMI) [6]. It is provided in a separate file in order to allow import and processing of the eODL metamodel by UML tools.
- Clause 1 provides an overview of how eODL is used by designers, implementers and managers of a distributed system. A concrete example of the use is given in Appendix I.

with:

- Annex D contains a software reference to the XML representation [12] of the eODL metamodel according to the XML meta interchange format (XMI) [6]. It is provided in a separate file in order to allow import and processing of the eODL metamodel by UML tools.
- Annex E contains mapping rules from technology independent eODL to technology specific CIDL [7].
- Appendix I provides an overview of how eODL is used by designers, implementers and managers of a distributed system. A concrete example of the use is given in Appendix I.

2) Update the Contents as follows

Add the following entries to the Contents paragraph before Appendix I. Modify the page number according to the new document.

3) Insert before Appendix I

Annex E

eODL to CIDL mapping

E.1 Introduction

The component based software development is an approach for modular and model-based software development. Supported by different mappings, it transforms **component models** (seen from different views like design and implementation) to the binary **software components**. The composition of software components takes place during execution time.

eODL is a language which offers concepts for **a technology independent model description** of the components in their life cycle from different viewpoints. Concepts like computational object, component, interface, module, signal, and data type are essential for the computational and implementation viewpoint. In addition, there are further concepts for the description of runtime environments and deployment of software components.

The **CCM** (CORBA Component Model [7]) is an OMG standard for a platform dependent framework. It provides a metamodel for the description of **technology dependent** CORBA components and the technology and runtime environment for components developed using that metamodel. CCM is based on mature CORBA technologies like the GIOP protocol and language bindings for implementation languages. The component model of CCM defines two kinds of component interactions. There is a RPC-like interaction with request/response and a signal-like one with events. For each of these interaction kinds components can declare the usage or the provision. For the notation of component implementation CCM uses the language CIDL.

To **bridge the gap** between technology-independent software component models given as eODL specifications technology-dependent models given as CIDL models, mappings are needed as a base for automated model transformations.

The OMG Component Implementation Definition Language (**CIDL**) is a language used to describe the structure and state of CORBA component implementations. Component-enable compilers generate implementation skeletons from CIDL definitions. Component builders extend these skeletons to create complete implementations.

This annex defines the rules for an eODL to CIDL mapping. These rules are verified by a compiler implementation.

E.2 Restricted mapping from eODL to CIDL

The definition of eODL is widely based on concepts defined by CORBA IDL 2.x [5]. Also the metamodel of eODL forms an extension of the CORBA metamodel. The adopted concepts are assigned to the *computational viewpoint* of eODL. Unfortunately the metamodel of CCM does not support the eODL concepts of the *deployment viewpoint* and the *target environment viewpoint*. This field is not defined by the MOF metamodel of CCM yet. There are only XML document types defined that are necessary for the final realization of the deployment architecture.

Conclusion: The eODL concepts concerning the *deployment viewpoint* and the *target environment viewpoint* are not mapped. The mapping rules should be extended as far as an according OMG standardization process will be finalized.

E.3 Mapping of eODL concepts which are CORBA concepts on CCM

Like the eODL metamodel, the CCM metamodel also extends the CORBA 2.x concepts. Therefore, for eODL concepts which derive from CORBA, the simplest mapping is chosen, **the identical mapping**. This allows the assignment of basic concepts from the eODL *computational viewpoint* like **data types, interfaces, operations and attributes** from the platform independent level to the platform specific level in a way which is the simplest one for the developer.

By using the CORBA metamodel as source as well as destination for the mapping, the occurrence of overlapping by defining the transformation rules is possible. Because of the identical mapping, this only happens when concepts of the CORBA metamodel are used in a context which is not derived from the CORBA metamodel.

E.4 Mapping of the computational viewpoint concepts

E.4.1 Signal

Signals are information carrier in eODL. They are transported during a signal-based interaction from the sender to the receiver.

Rule 1: For each **SignalDef** in eODL an **EventDef** in CCM with the same name is created. The associated names and data types in a **CarryField** in eODL are mapped onto **ValueMemberDef** elements in CCM, which are contained within the **EventDef**. All created **ValueMemberDef** have public visibility (isPublicMember==true).

Example:

```
signal Sig {
  long l;
};
```

Is mapped to CIDL,

```
eventtype Sig {
  public long l;
};
```

E.4.2 Consume and Produce

The interaction elements consume and produce of eODL are supposed to define the signal-based interaction within an interface. Even though signal-based interaction exists in CCM (***EventDef***), it is not allowed to be part of an interface. CCM defines such an interaction only as a direct part of a component definition. Again this is not allowed in eODL: only attributes are permitted. A complete prohibition of consume and produce in the eODL model would prevent a signal-based interaction. Therefore a replacement construction is defined, which does increase the complexity of the mapping but at least allows signal-based interaction. In CCM the definition of a signal (***EventDef***) forms a definition of an interface for signal exchange. When defining component ports, they are handled as own ports. So for each signal-based interaction element of eODL, a separate port is defined at the component.

Rule 2: Elements of type ***ConsumeDef*** and ***ProduceDef*** in eODL are not mapped themselves but are handled by the rules for ports.

Example:

```
signal Sig;
interface A {
  consumes Sig c;
  produces Sig p;
};
```

Is mapped to CIDL,

*where the signal-based interaction elements from the **EnhancedInterfaceDef** are removed. Only signals are reflected in CCM as **EventDef**.*

```
eventtype Sig {
};
interface A {
};
```

E.4.3 Media, sink and source

Operational and signal-based interactions are supported by CCM. A Stream-based interaction **is not reflected.** There are efforts to extend CCM by those concepts but these are not part of the standard.

Therefore, the mapping will not transform the model elements of these concepts from eODL to CCM.

E.4.4 CO-type, supports and requires

Both eODL and CCM are extensions of CORBA concepts and introduce the concept of a component. In eODL this concept is called CO-type, whereas CCM calls it component. So a CO-type is mapped onto a component. Interactions of the CO-type are only allowed via ports because this interaction variant also exists in CCM.

Rule 3: For each ***COTypeDef*** in eODL a ***ComponentDef*** in CCM with the same name is created. If the ***COTypeDef*** B specializes ***COTypeDef*** A in eODL, then the corresponding ***ComponentDef*** B specializes ***ComponentDef*** A in CCM. The relations ***supports*** and ***requires*** are not mapped. Multiple inheritance of ***COTypeDef*** in eODL is not permitted.

Example:

```
CO A {
};
CO B {
};
```

The CO-types of eODL are mapped onto components in CCM while the inheritance relation is preserved.

```
component A {
};
component B : A {
};
```

E.4.5 Home (HomeDef)

In CCM a further concept exists that is related closely to CCM component. The concept home is used for managing the components during runtime. A home provides a facility for creating instances of the components. That is why a component definition without a home is incomplete. So the mapping creates for each CO-type also a *home* in CCM.

Rule 4: For each ***COTypeDef*** in eODL, a ***HomeDef*** in CCM is created constructing its name by concatenating the name of the ***COTypeDef*** and "**`_Home`**". The resulting ***ComponentDef*** and ***HomeDef*** take part in the ***Component_Home*** association. If a ***COTypeDef*** B specializes ***COTypeDef*** A in eODL then the corresponding ***HomeDef*** B specializes ***HomeDef*** A in CCM.

Example:

```
CO A {
};
CO B : A {
};
```

The mapping creates for each CO-type a component type in CCM as well as a home.

```
component A {
};
home A_Home manages A {};
component B : A {
};
```

```
home B_Home : A_Home manages B {};
```

E.4.6 Provide and used port

The concepts ***ProvidePortDef*** and ***UsedPortDef*** in eODL are used to define the interface between the CO-type and the environment. In CCM the corresponding concept is port (***ComponentFeature***), which occurs in different specializations. There is a port for the realization and usage of an interface in the context of CORBA as well as one for signal-based interaction.

Rule 5: For each ***ProvidePortDef*** in eODL within a ***COTypeDef***, a ***ProvidesDef*** of the corresponding ***ComponentDef*** in CCM with the same name is created. The referenced ***InterfaceDef*** according to the ***ProvidePortDef*** is part of the ***provides*** association of the ***ProvidesDef***. The ***ProvidesDef*** has the role ***facet*** in respect to the ***ComponentDef*** within the aggregation ***Component_Facet***. Multiple ports (*multiple==true*) are not allowed.

Example:

```
interface A {
};
CO C {
  provides A the_a;
};
```

Provided interface types of the CO-type without signal-based interaction are mapped onto simple ports.

```
interface A {
};
component C {
  provides A the_a;
};
```

Rule 6: For each ***UsedPortDef*** in eODL within a ***COTypeDef***, a ***UsesDef*** of the corresponding ***ComponentDef*** in CCM with the same name is created. The referenced ***InterfaceDef*** according to the ***UsedPortDef*** is part of the ***uses*** association of the ***UsesDef***. The ***UsesDef*** has the role ***receptacle*** in respect to the ***ComponentDef*** within the aggregation ***Component_Receptacle***. Multiple ports in eODL (*multiple==true*) are mapped onto multiple ports (*multiple==true*).

Example:

```
interface A {
};
CO C {
  uses multiple A the_a;
};
```

*In contrast to mapping of the **ProvidesDef**, multiple ports are allowed as used ports.*

```
interface A {
};
component C {
  uses multiple A the_a;
};
```

E.4.7 Produce and consume port

Because the interaction elements were removed from the interfaces due to the mapping (*Rule 2*), additional rules for the mapping of signal-based interaction with port in interfaces have to be defined.

Rule 7: For each ***ProduceDef*** in eODL within an ***InterfaceDef*** that is referenced by a ***ProvidePortDef*** of a ***COTypeDef,*** a ***PublishesDef*** is created with a name constructed of the concatenation of the name of the ***ProvidedPortDef*** with "_" and the name of the ***ProduceDef***. The ***PublishesDef*** has the role ***publishes*** in respect to the ***ComponentDef*** within the aggregation ***Component_Publishes***.

Example:

```
signal Sig;
interface A {
  produce Sig p;
};
CO C {
  provides A the_a;
};
```

The usage of interfaces with signal-based interaction of port always results in a separated port because each signal-based interaction has to be mapped onto a separate port.

```
eventtype Sig {
};
interface A {
};
component C {
  provides A the_a;
  publishes Sig the_a_p;
};
```

Rule 8: For each ***ConsumeDef*** in eODL within an ***InterfaceDef*** that is referenced by a ***ProvidePortDef*** of a ***COTypeDef,*** a ***ConsumesDef*** in CCM is created with a name constructed of the concatenation of the name of the ***ProvidedPortDef*** with "_" and the name of the ***ConsumeDef***. The ***ConsumesDef*** has the role ***consumes*** in respect to the ***ComponentDef*** within the aggregation ***Component_Consumes***.

Example:

```
signal Sig;
interface A {
  consume Sig s;
};

CO C {
  provides A the_a;
};
```

Different to the previous example, the direction of the signal-based interaction has changed, that is why the sending port is now a receiving port.

```
eventtype Sig {
};
interface A {
  consume Sig c;
};
component C {
  provides A the_a;
  consumes Sig the_a_c;
};
```

Rule 9: For each ***ProduceDef*** in eODL within an ***InterfaceDef*** that is referenced by a ***UsedPortDef*** of a ***COTypeDef,*** a ***ConsumesDef*** in CCM is created with a name constructed of the concatenation of the name of the ***ProvidedPortDef*** with "_" and the name of the ***ConsumeDef***. The ***ConsumesDef*** has the role ***consumes*** in respect to the ***ComponentDef*** within the aggregation ***Component_Consumes***.

Example:

```
signal Sig;
interface A {
  produce Sig p;
};

CO C {
  use A the_a;
};
```

The mapping is the same as in Rule 7 except for the mapping of a used port.

```
eventtype Sig {
};
interface A {
};
component C {
   uses A the_a;
  consumes Sig the_a_p;
};
```

Rule 10: For each ***ConsumeDef*** in eODL within an ***InterfaceDef*** that is referenced by a ***UsedPortDef*** of a ***COTypeDef,*** a ***PublishesDef*** in CCM is created with a name constructed of the concatenation of the name of the ***UsedPortDef*** with "_" and the name of the ***ConsumeDef***. The ***ConsumesDef*** has the role ***publishes*** in respect to the ***ComponentDef*** within the aggregation ***Component_Publishes***.

Example:

```
signal Sig;
interface A {
  consume Sig s;
};
CO C {
  use A the_a;
};
```

Compared with the previous example, the direction of the signal-based interaction has changed therefore the sending port is now a receiving port.

```
eventtype Sig {
};
interface A {
  consume Sig c;
};
component C {
  uses A the_a;
  publishes Sig the_a_c;
};
```

E.4.8 Attribute

CO-types in eODL can contain only ***AttributeDef*** as interaction elements. Also the corresponding concept in CCM can contain ***AttributeDef***, so ***AttributeDef*** of CO-types are mapped onto ***AttributeDef*** of component types.

Rule 11: ***COTypeDef*** as specialization of ***InterfaceDef*** with the constraints defined by ITU-T Rec. Z.130 is allowed to contain ***AttributeDef***. For each ***AttributeDef*** in a ***COTypeDef***, an ***AttributeDef*** in CCM is created within the corresponding ***ComponentDef*** in CCM. Name, association and attributes of the ***AttributeDef*** are preserved according to E.1.

Example:

```
CO C {
  readonly attribute long l;
};
```

Actually, the mapping of attributes is an identical one but it is used outside the context of CORBA.

```
Component C {
  readonly attribute long l;
};
```

E.5 Mapping of implementation viewpoint concepts

The concepts of eODLs implementation viewpoint are describing the relation between the interaction elements provided by the CO-type interface types and the realizations through artefacts of the implementation language. In doing so, elements of the implementation language of the component technology are modelled by artefacts. In the context of object oriented implementation languages there are usually classes. eODL is less restrictive considering the relation of interaction elements and artefacts. Especially there is no constraint given regarding the grouping of interaction elements and interfaces. CCM also defines concepts for specifying the structure of component realization. For realizing this, the concept ***ComponentImplDef*** is used. Because each component has a defined home, there is a concept ***HomeImplDef*** for its realization. CCM also defines the concept ***SegmentDef*** for further specification of the substructure. Using the ***SegmentDef***, an artefact is defined that realizes the provided interfaces of the component. That means CCM allows only relations between the interaction elements and artefacts according to their grouping in interfaces (see Figure E.1). Furthermore the assignment of the component used interfaces to ***SegmentDef*** is forbidden. Connecting signal-based interaction elements to ***SegmentDef*** in case of sending is not allowed as well as in case of receiving. At least the assignment of receiving signals should be possible in CCM.

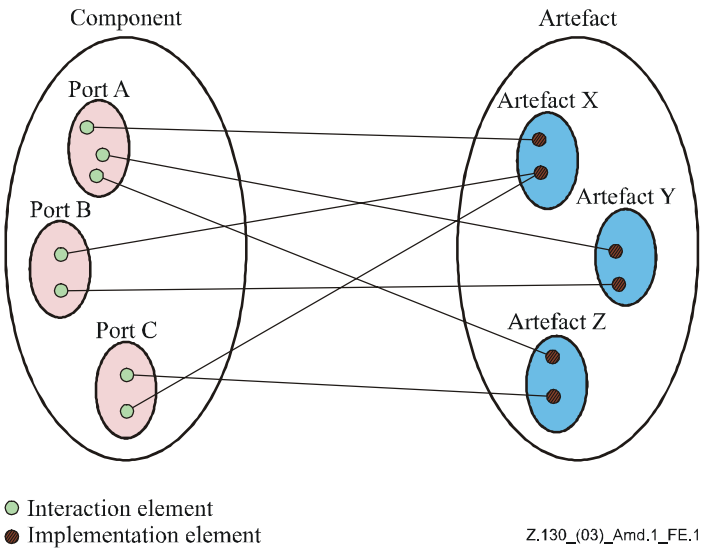

Figure E.1/Z.130 – Assignment of interaction elements and artefacts

E.5.1 Artefact

eODL uses the concepts ***ArtifactDef*** and ***ImplementationalElementDef*** for describing the structure of a CO-type realization and the assignment of interaction elements to the artefacts. CCM specifies the concept ***SegmentDef*** for this assignment. Therefore artefacts are mapped onto ***SegmentDef***. eODL defines for each artefact one of the following instantiation patterns:

ARTIFACT_PER_REQUEST, ***ARTIFACT_POOL***, ***SINGLETON*** or ***USER_DEFINED***.

They are describing the time of instantiation of an artefact. The closest concepts in CCM are the component types categories ***PROCESS***, ***ENTITY***, ***SESSION*** and ***SERVICE***. On one hand, they are only referring to whole components realization when a separation for specific ***SegmentDef*** is not given and, on the other hand, no unique assignments between instantiation pattern and the component types categories can be found (see 5.4.1 and CORBA Components, section 4.1.4).

Rule 12: For each ***ArtifactDef*** and the contained ***ImplementationalElementDef*** in eODL, a ***SegmentDef*** with the same name is created in CCM. The ***ImplementationElementDef*** contained within the ***ArtifactDef*** are only allowed to be related with interaction elements of the type ***OperationDef*** and ***AttributeDef*** for interface types (*Case == supply*) provided via ports. All ***ImplementationElementDef*** elements within an ***ArtifactDef*** have to cover all interaction elements of an ***InterfaceDef***. The association ***implementedBy*** in CCM then contains the ***SegmentDef*** and all ***ProvidesDef*** elements according to the different ***InterfaceDef*** elements. The association ***implemented_by*** of eODL is mapped onto the association ***segments*** of CCM according to the corresponding created elements.

For all ***ArtifactDef*** elements in eODL, which are realizing the same CO-type, a ***ComponentImplDef*** in CCM is created. Its name is formed by concatenating the name of the ***COTypeDef*** and "**`Impl`**". The component implementations category is session (*category == SESSION*). The ***ComponentImplDef*** and the ***ComponentDef*** according to the ***COTypeDef*** take part in the ***implemented_by*** relation.

For each created ***HomeDef*** according to the ***COTypeDef*** a ***HomeImplDef*** is created with the name formed by concatenating the ***HomeDef*** name and "**`Impl`**". The ***HomeImplDef*** is associated to the ***HomeDef*** by the ***implements*** relation and to the ***ComponentImplDef*** by the ***manages*** relation.

Example:

```
interface A {
  void op();
};

CO C {
  provides A the_a;
};
artefact AImpl {
  op implements supply A::op;
};
```

In eODL there is no separate concept for a CO-type implementation; nevertheless, CCM needs a ComponentImplDef. For the implicitly created HomeDef also a HomeImplDef has to be created.

```
interface A {
  void op();
};
component C {
  provides A the_a;
};

home C_Home manages C {};
composition session CImpl {
  home executor C_HomeImpl {
    implements C_Home;
    manages CSessionImpl {
      segment AImpl {
        provides facet the_a;
      };
    };
  };
};
```

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	**Languages and general software aspects for telecommunication systems**

Printed in Switzerland
Geneva, 2006